Федеральное государственное автономное
образовательное учреждение
высшего образования
"СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ"

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____А. И. Легалов
"___"_____20____г.

БАКАЛАВРСКАЯ РАБОТА

09.03.63 Информатика и вычислительная техника

Электронная система учета посещений

Руководитель	_____ *подпись*	_____ *дата*	к.т.н., доцент каф. *должность, ученая степень*	С.Н. Титовский
Выпускник	_____ *подпись*	_____ *дата*		О.С. Салтынюк
Консультант	_____ *подпись*	_____ *дата*	ст. преподаватель *должность, ученая степень*	В.А. Поваляев
Нормоконтролёр	_____ *подпись*	_____ *дата*		В.И. Иванов

Красноярск 2016

РЕФЕРАТ

Выпускная квалификационная работа (ВКР) на тему "Электронная система учета посещений " содержит 56 страниц текстового документа, 9 рисунков, 12 источников литературы, презентацию 14 слайдов.

СЧИТЫВАЮЩЕЕ УСТРОЙСТВО, METEOR, МИКРОСЕРВЕСЫ, БАЗЫ ДАННЫХ, PYTHON, АВТОМАТИЧЕСКОЕ ВЫСТАВЛЕНИЕ ПРОПУСКОВ, RASPBERRY PI.

Цель работы: Создать электронную систему учета посещаемости студентов на занятиях в институте.

Задачи: Спроектировать считывающее устройство; создать сайт для обработки данных; организовать общую работу системы и отправки данных на сайт e.sfu-kras.ru

Эта система очень актуальна так как аналогов в среде образования ей нет. Также она очень практична и эффективна так как помогает с экономить много времени и исключить ошибки которые появляются в результате человеческого фактора.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В Сибирском Федеральном Университете, как и во всех учебных заведениях отмечают посещение студентами занятий. Это занимает очень много времени особенно на лекционных занятиях, так как лекция проводится одновременно для нескольких групп.

Поэтому цель диплома автоматизировать учет посещений.

В следствии того что у студентов нашего института вместо студенческих билетов есть карты с RFID меткой, по которой они проходят в институт. И также вузе есть система электронного обучения выражается она в виде сайта с электронными курсами, которая содержит дополнительную информацию о курсах и также туда необходима вносить информацию о посещениях.

Были поставлены задачи: спроектировать считывающее устройство; создать сайт для обработки данных; создать программу для обработки информации о посещениях, и отправки данных на сайт e.sfu-kras.ru.

Электронная система учета посещений не я является новой разработкой она используется на некоторых предприятиях для учета времени в которое приходят сотрудники и т. д., но однако, мне не встречалась разработка которая бы учитывала посещения в учебных заведениях. Моя электронная система учета предназначена для автоматизации учета посещений в учебных заведениях, в частности, в нашем институте. Она отправляет информацию о посещениях на сайте e.sfu-kras.ru при условии, что студент, придя на занятия, отметился с помощью электронной карты студента на считывающем устройстве у двери кабинета.

1 Анализ задач

Для осуществления поставленных задачь понадобилось множество компонентов.

Возле каждого кабинета в институте должно стоять считывающее устройство, подключённое к микрокомпьютеру, который осуществляет передачу данных на сервер.

Частично эта задача уже решена, тем что у каждого кабинета стоит считывающее устройство, которое в данный момент работает как электронный ключ.

Так как нам нужно хранить информацию о студентах, которые пришли на занятия и о расписание занятий в аудиториях. Для этого необходима база данных.

Также нам необходима возможность редактировать данные о студентах и расписании.

Это будет осуществляется с помощью сайта, который будет находиться во внутренней сети института. В этом случаи доступ к нему будет с любого компьютера института и это будет достаточно безопасно, так как является внутренней сетью.

Еще необходимо заносить информацию о посещениях на сайт e.sfu-kras.ru.

Для этого необходима программа, которая будет обрабатывать всю приходящую информацию и отправлять запросы на сайт для заполнения таблиц посещаемости.

1.1 Считывающее устройство

1.1.1 Модель устройства и причины выбора

Было взято устройство модели RFID RC522 13.56MHz (рисунок 1). Этот тип считывающего устройства был выбран, так как в СФУ в качестве электронных ключей у каждого кабинет используется тот же стандарт считывающих устройств.

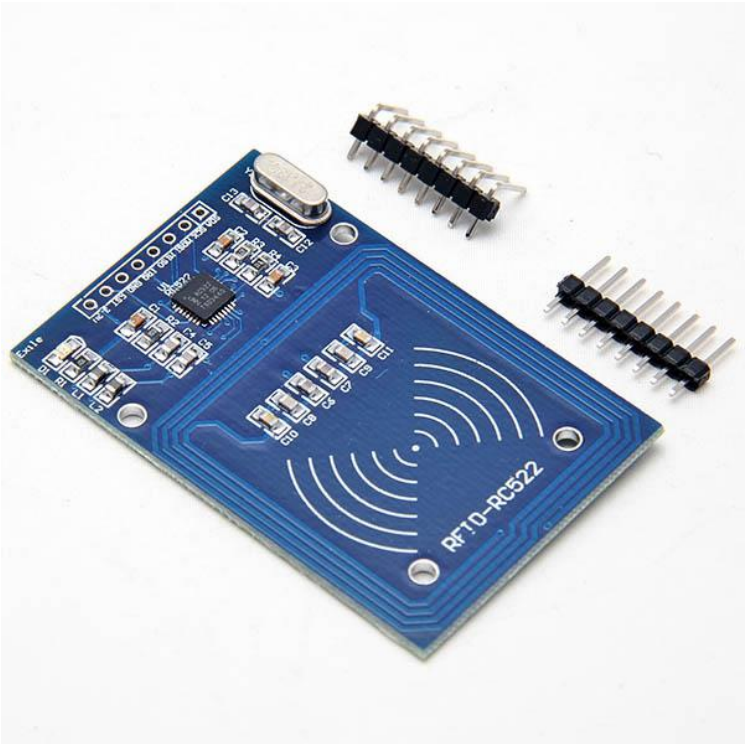


Рисунок 1 – Изображение RFID RC522

Существует много разных видов и технологий идентификации, например, смарт-карты магнитные карты и т. д. Мы же решили остановиться на технологии Технология радиочастотной идентификации Radio Frequency Identification – RFID.

RFID — это современная технология идентификации, предоставляющая существенно больше возможностей по сравнению с традиционными системами маркировки.

1.1.2 Компоненты rfid-системы

Метки (tag) RFID — устройства, способные хранить и передавать данные. В памяти меток содержится их уникальный идентификационный код. Некоторые метки имеют перезаписываемую память.

Считыватели (reader) — приборы, которые читают информацию с меток и записывают в них данные. Эти устройства могут быть как постоянно подключенными к учетной системе, так и работать автономно.

RFID-метка представляет собой миниатюрное запоминающее устройство. Она состоит из микрочипа, который хранит информацию, и антенны, с помощью которой метка эти данные передает и получает. Иногда RFID-метка имеет собственный источник питания (такие метки называют активными), но большинство меток его лишены (эти метки называют пассивными).

В памяти RFID-метки хранится ее собственный уникальный номер и пользовательская информация. Когда метка попадает в зону регистрации, эта информация принимается считывателем, специальным прибором, способным читать и записывать информацию в метках.

1.1.4 Преимущества радиочастотной идентификации

– Возможность перезаписи. Данные RFID-метки могут перезаписываться и дополняться много раз, тогда как данные на штрих-коде не могут быть изменены — они записываются сразу при печати;

– Отсутствие необходимости в прямой видимости. RFID-reader не требуется прямая видимость метки, чтобы считать ее данные. Взаимная ориентация метки и считывателя часто не играет роли. Метки могут читаться через упаковку, что делает возможным их скрытое размещение.

Для чтения данных метке достаточно попасть в зону регистрации, в том числе при перемещении через нее на достаточно большой скорости. Напротив, устройству считывания штрих-кода всегда необходима прямая видимость штрих-кода для его чтения;

– Большее расстояние чтения. RFID-метка может считываться на значительно большем расстоянии, чем штрих-код. В зависимости от модели метки и считывателя радиус считывания может составлять до нескольких десятков метров;

– Больший объем хранения данных. RFID-метка может хранить значительно больше информации, чем штрих-код. До 10 000 байт могут храниться на микросхеме площадью в 1 квадратный сантиметр, в то время как штриховые коды могут вместить 100 байт (знаков) информации, для воспроизведения которых понадобится площадь размером с лист формата А4;

– Поддержка чтения нескольких меток. Промышленные ридеры могут одновременно считывать несколько десятков RFID-меток в секунду, используя так называемую анти коллизионную функцию. Устройство считывания штрих кода, однако, может единовременно сканировать только один штрих-код;

– Считывание данных метки при любом ее расположении. В целях обеспечения автоматического считывания штрихового кода, комитетами по стандартам (в том числе EAN International) разработаны правила размещения штрих-меток на товарной и транспортной упаковке. К радиочастотным меткам эти требования не относятся. Единственное условие — нахождение метки в зоне действия сканера;

1.1.5 Что нужно помнить при внедрении RFID

При работе с радиочастотной идентификацией необходимо учитывать некоторые ограничения. К ним относятся: относительно высокая стоимость,

невозможность размещения под металлическими и экранирующими поверхностями, взаимные коллизии, подверженность помехам в виде электромагнитных полей.

– Относительно высокая стоимость RFID-меток. Стоимость пассивной радиочастотной метки составляет от 0,15 доллара (при приобретении свыше 1 000 000 шт.) до 3 долларов (при приобретении 1 шт.). В случае с метками защищенного исполнения (или на металл) эта цена может достигать 7 и более долларов. Таким образом, стоимость RFID-меток превышает стоимость этикеток со штриховым кодом. Исходя из этого, использование радиочастотных меток целесообразно для защиты дорогих товаров от краж или для обеспечения сохранности изделий, переданных на гарантийное обслуживание. В сфере логистики и транспортировки грузов стоимость радиочастотной метки оказывается совершенно незначительной по сравнению со стоимостью содержимого контейнера, поэтому использование радиочастотных меток совершенно оправдано на упаковочных ящиках, паллетах и контейнерах;

– Возможное экранирование при размещении на металлических поверхностях. RFID-метки подвержены влиянию металла (это касается упаковок определенного вида — металлических контейнеров, иногда даже некоторых типов упаковки жидких пищевых продуктов, запечатанных фольгой). Это вовсе не исключает применение RFID, но приводит или к необходимости использования более дорогих меток, разработанных специально для установки на металлические поверхности, или к нестандартным способам закрепления меток на объекте;

– Подверженность систем радиочастотной идентификации помехам в виде электромагнитных полей от включенного оборудования, излучающего радиопомехи в диапазоне частот, используемом для работы RFID-системой. Необходимо тщательно проанализировать условия, в которых система RFID будет эксплуатироваться. Для систем UHF диапазона 868-869 МГц это практически не актуально (в этом диапазоне никакие другие приборы

не работают), но низкочастотные метки, работающие на частоте 125 КГц подобному влиянию подвержены;

1.2 Raspberry Pi

Raspberry Pi — одноплатный компьютер размером с банковскую карту, изначально разработанный как бюджетная система для обучения информатике, впоследствии получивший намного более широкое применение и популярность, чем ожидали его авторы. Разрабатывается Raspberry Pi Foundation.

1.2.1 История

В мае 2011 года Дэвид Брэбен представил первый концепт Raspberry Pi размером с USB-флеш-накопитель.

В конце июля 2011 года была закончена и отправлена в производство альфа-версия платы, а уже 12 августа Raspberry Pi Foundation получила первую партию устройств. Альфа-версия компьютера содержала некоторые тестовые функции и дорогие детали, которые убрали из финальной версии. Также конечная версия платы на 20% меньше и состоит из четырёх слоёв, а не из шести.

10 января 2012 года компания объявила о начале производства первой партии из 10 тысяч плат модели «B»

29 февраля 2012 года началась продажа плат модели «B»

16 июля 2012 года разработчики проекта сообщили о снятии ограничений на заказ устройств Raspberry Pi «B»

14 декабря 2012 года Raspberry Pi «A» запущена в производство.

14 июля 2014 года разработчики проекта выпустили третью версию Raspberry Pi «B+».

2 февраля 2015 года разработчики проекта выпустили четвертую версию Raspberry Pi «2B».

26 ноября 2015 года разработчики проекта выпустили новый микрокомпьютер Raspberry Pi Zero. Основные отличия – цена в пять долларов и несмонтированный разъём GPIO.

1.2.2 Технические подробности

Raspberry Pi выпускается в нескольких комплектациях: модель «A», модель «B», модель «B+» и модель «2 B». Первые три версии оснащены ARM11 процессором Broadcom BCM2835 с тактовой частотой 700 МГц и модулем оперативной памяти на 256МБ/512МБ, размещенными по технологии «package-on-package» непосредственно на процессоре. Модель «2B» оснащается процессором с 4 ядрами Cortex-A7 с частотой 1ГГц и оперативной памятью размером 1ГБ. Модель «A» оснащается одним USB 2.0 портом, модель «B» двумя, а модели «B+» и «2 B» — четырьмя. Также в моделях «B», «B+» и «2 B» присутствует порт Ethernet. Помимо основного ядра, BCM2835 включает в себя графическое ядро с поддержкой OpenGL ES 2.0, аппаратного ускорения и FullHD-видео и DSP-ядро. Одной из особенностей является отсутствие часов реального времени.

Вывод видеосигнала возможен через композитный разъём RCA или через цифровой HDMI-интерфейс. В версии «B+» и «2B» вывод возможен через аудиоразьем 3,5. Корневая файловая система, образ ядра и пользовательские файлы размещаются на карте памяти SD, MMC, microSD (только в модели «B+») или SDIO.

Одной из самых интересных особенностей Raspberry Pi является наличие портов GPIO (general purpose input/output). Благодаря этому "малиновый" компьютер можно использовать для управления различными

устройствами. В модели «B» платы присутствуют 26 портов, а в модели «B+» и «2 B» - 40 портов GPIO.

1.2.3 Программное обеспечение

Raspberry Pi работает в основном на операционных системах, основанных на Linux ядре. Запуск Windows возможен благодаря средствам виртуализации
таким, как XenDesktop. ARM11 основан на 6 версии ARM, на котором несколько популярных версий Linux больше не запускаются. Для установки операционных систем существует инструмент NOOBS.

1.2.4 Внешний вид и выбор модели.

Компьютер распространяется полностью собранным на четырёхслойной печатной плате размером с банковскую карту. В стандартный комплект поставки входит только сама плата. Корпус, блок питания, флеш-карту необходимо заказывать отдельно.

На данном этапе разработки проекта мы использовали Raspberry Pi модель

«B» так как он была у нас в наличии и является самой дешёвой моделью из списка с необходимыми для нас компанентами.

Так выглядит плата «B» на которой мы разрабатывали наш проект (рисунок 2).

Рисунок 2 – Изображение Raspberry Pi «B»

1.3 Meteor

Одна из задач — то хранение и редактирование данных в этом вопросе первичным была проблема редактировать данные. Был выбран достаточно удобный и простой путь — это сайт на внутренней сети института, потому что он всегда доступен с любого кабинета и для пользование не нужно устанавливать программу и потом ее обновлять и т. д. Для разработки сайта нам, конечно, нужны средства или языки для его написания с frontend разработкой нет смысла что-то придумывать и есть один путь — это HTML для разметки, JavaScript для работы кнопок и вкладок, CSS для верстки, и также мне очень помог Bootstrap. Для разработки backend или сердце (ядра) нашего сайта был взят веб-фраемворк Meteor. Конечно, есть ещё много всяких возможностей и языков таких как PHP и Python**.** Meteor — веб-платформа на языке JavaScript, предназначенная для разработки Web-приложений реального времени. Для связи с современными браузерами используется протокол Distributed Data Protocol (DDP), поддерживаемый с помощью WebSocket'ов либо, если поддержки веб-сокетов и DDP нет — AJAX.

1.3.1 Внутреннее строение Meteor

Meteor - это платформа для создания так называемых real-time web apps - современных веб-приложений. По сути, Meteor - это слой между интерфейсом вашего приложения и его базой данных, который следит за их синхронизацией.

Поскольку фреймворк построен на основе Node.js, то JavaScript используется как на клиенте, так и на сервере. И более того, Meteor позволяет использовать один и тот же код и на клиенте, и на сервере

В результате всего этого мы получаем очень мощную, и при этом простую в использовании платформу, так как большинство стандартных рутин и трудностей создания веб-приложений уже реализованы из коробки.

1.3.2 Особенности

Код Meteor работает поверх node.js (однако он не придерживается принятой в node.js асинхронной модели, что может затруднить интеграцию

node.js и meteor-приложений).

Ядром Meteor является протокол DDP. Он предназначен для работы с коллекциями JSON-документов, позволяя легко создавать, обновлять, удалять, запрашивать и просматривать их. По умолчанию в качестве хранилища таких документов используется MongoDB.

Одна из важнейших особенностей платформы состоит в том, что она позволяет использовать один и тот же код как на стороне сервера, так и на стороне клиента. Между сервером и клиентом, как правило, передаются данные, а не HTML-код.

1.3.3 Преимуществ

В отличие от других фреймворков, он позволяет создать собственное real-time веб-приложение и выложить его в Интернете в течение всего лишь нескольких часов.

И также, если вы когда-нибудь занимались front-end разработкой, то вы уже знакомы с JavaScript и не нужно изучать новый язык.

Еще одна особенность и удобство это обновление изменений в реальном времени практически мгновенно если вы что-то исправили у себя в браузере удалили какую-то информацию или отредактировали ее то через несколько минут она появится у другого пользователя и ему даже не придется обновлять страницу.

На протяжении многих лет мы мирились с тем, что общаться с вебсайтом можно лишь средствами отдельных, коротких запросов.

Но Meteor является представителем новой волны веб-фреймворков, которые бросают вызов устоявшемуся несовершенному порядку, внедряя современные концепции real-time web и reactive programmi.

1.4 База данных

Ну и так как разработчики Meteor предлагают использовать нам базу данных MongoDB мы конечно с ними соглашаемся так как она отвечает всем нашим требованиям.

MongoDB — документно-ориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Написана на языке C++.

1.4.1 Возможности

Основные возможности:

– Документно-ориентированное хранение (JSON-подобная схема данных);

– Javascript как язык для формирования запросов;

– Динамические запросы;

– Поддержка индексов;

– Профилирование запросов;

– Атомарная операция «Нашел и обновил»;

– Эффективное хранение двоичных данных больших объёмов, например, фото и видео;

– Журналирование операций, модифицирующих данные в базе данных;

– Поддержка отказоустойчивости и масштабируемости: асинхронная репликация, набор реплик и распределения базы данных на узлы;

– Может работать в соответствии с парадигмой MapReduce;

– Полнотекстовый поиск, в том числе на русском языке, с поддержкой морфологии;

1.4.2 Архитектура

СУБД управляет наборами JSON-подобных документов, хранимых в двоичном виде в формате BSON. Хранение и поиск файлов в MongoDB происходит благодаря вызовам протокола GridFS. Подобно другим документоориентированным СУБД (CouchDB и др.), MongoDB не является реляционной СУБД.

В Mongo:

Нет такого понятия, как «транзакция». Атомарность гарантируется только на уровне целого документа, то есть частичного обновления документа произойти не может.

Отсутствует понятие «изоляции». Любые данные, которые считываются одним клиентом, могут параллельно изменяться другим клиентом.

В MongoDB реализована асинхронная репликация в конфигурации «ведущий — ведомый» (англ. *master — slave*), основанная на передаче журнала изменений с ведущего узла на ведомые. Поддерживается автоматическое восстановление в случае выхода из строя ведущего узла. Серверы с запущенным процессом mongod должны образовать кворум, чтобы произошло автоматическое определение нового ведущего узла. Таким образом, если не используется специальный процесс-арбитр (процесс mongod, только участвующий в установке кворума, но не хранящий никаких данных), количество запущенных реплик должно быть нечётным.

Имеется подробная и качественная документация, большое число примеров и драйверов под популярные языки Java, JavaScript, Node.js, C++, C#, PHP, Python, Perl,Ruby, Grails&Groovy.

Заявляется, что релиз MongoDB 1.0.0 готов к использованию в производстве как в качестве единичного мастера, так и в связках «ведущий — ведомые». Код этого релиза достаточно стабилен и проверен в промышленной эксплуатации на протяжении 1,5 лет. По возможности MongoDB должна быть развернута минимум на двух серверах, используя репликацию Master/Slave. Это обеспечивает наличие актуальных данных при выходе из строя одной из СУБД.

Технологии развиваются семимильными шагами. Список новых технологий и методологий постоянно растёт. Однако, я всегда придерживался мнения, что фундаментальные технологии, используемые программистами, развиваются не столь стремительно. Можно долгое время обладать актуальными знаниями, не пополняя их. Однако зачастую устоявшиеся технологии заменяются другими с потрясающей скоростью. Внезапные скачки разработок иногда ставят под угрозу устоявшиеся старые технологии.

Яркий пример того — прогресс NoSQL-технологий, приходящих на замену давно известным реляционным базам данных. Вчера ещё веб

базировался на нескольких известных СУРБД, однако уже сегодня появилось около пяти NoSQL-решений, достойно зарекомендовавших себя.

Несмотря на скачкообразность таких изменений, на деле могут понадобиться годы, чтобы они вошли в общепринятую практику. Начальный энтузиазм, как правило, охватывает небольшое число разработчиков и компаний. Решения оттачиваются, извлекаются уроки, — и, видя, что новая технология развивается, остальные пробуют применять её для своих нужд. Опять же, это касается сферы NoSQL, где множество технологий являются не столько прямой заменой более традиционным механизмам хранения, сколько являются решениями специальных проблем, в дополнение к тому, что можно ожидать от традиционных систем.

Принимая во внимание всё вышеизложенное, мы должны уяснить, чем же является NoSQL. Это широкий термин, который означает разное для разных людей. Лично я использую его в широком смысле, чтобы обозначить систему, участвующую в хранении данных. С другой стороны NoSQL для меня означает

убеждённость в том, что задача хранения данных не возлагается на одну большую систему. В то время, как производители большинства баз данных исторически пытались позиционировать свой софт, как решение «всё в одном», NoSQL стремится к меньшему уровню ответственности — когда для определённых задач может быть выбран такой инструмент, который бы решал именно эту задачу наилучшим образом. К примеру, ваш NoSQL-стек может эффективно использовать реляционные базы данных, как например MySQL, однако он также может включать в себя Redis — для организации хранения записей key-value или Hadoop — для интенсивной обработки данных. Проще говоря, NoSQL — это открытая технология, состоящая из альтернативных, существующих и дополнительных шаблонов управления данными.

Удивительно, но MongoDB подходит под все эти определения. Как документ-ориентированная СУБД, Mongo — это довольно-таки обобщённое

NoSQL решение. Её можно рассматривать, как альтернативу реляционным СУБД. Подобно реляционным СУБД, она также может выигрышно дополняться более специализированными NoSQL решениями.

1.5 Python

Python — высокоуровневый язык программирования, построенный на идея императивного, объектно-ориентированного и функционального программирования. Язык создан Гвидо ван Россумом в 1989 году и с тех пор непрерывно совершенствуется.

Преимущества Python:

– Открытая разработка;

– Язык довольно прост в изучении, особенно на начальном этапе;

– Особенности синтаксиса стимулируют программиста писать хорошо читаемый код;

– Предоставляет средства быстрого прототипирование и динамической семантики;

– Имеет большое сообщество, позитивно настроенное по отношению к новичкам;

– Множество полезных библиотек и расширений языка можно легко использовать в своих проектах благодаря предельно унифицированному механизму импорта и программным интерфейсам;

– Механизмы модульности хорошо продуманы и могут быть легко использованы;

– Абсолютно всё в Python является объектами в смысле ООП, но при этом объектный подход не навязывается программисту.

1.6 HTML

HTML — это язык гипертекстовой разметки, с помощью которого верстальщики создают структуру web-страниц, приложений и email-писем.

HTML формирует структуру веб-страницы посредством заголовков, списков и других подобных элементов, начиная от начала страницы — хедера(header), и до ее конца — футера(footer).

Структура web-страницы должна быть прописана на web-основе с помощью определенных команд.

В данном языке оперирует понятие тег. Тег это своего рода указатель на какую-либо информацию. Теги бывают парные и одиночные. Для парного тега необходимо определить начало() и конец(), для одиночного же тега присуще не закрытие формы.

Теги имеют так же еще и параметры, которыми можно указать размер объекта: `style="width: 445px; height: 445px;"`, шрифт текста: `style="font-family: 'Fjalla One', sans-serif;"` и т.д.

Свободные места внутри тегов называют контейнерами, теги определяют действие только для содержимого своего контейнера.

Простейший html-документ выглядит следующим образом:

```
<!DOCTYPE html>
<html>
<head>
<title>Заголовок</title>
</head>
<body>
<h1>Тут будет размещен заголовок</h1>
<p>Первый абзац</p>
</body>
</html>
```

Представленный небольшой вариант простого HTML-кода, содержит несколько элементов, состоящих, в свою очередь, из таких тэгов как:

– <head></head> — открывающий и закрывающий тэг, предназначен для хранения элементов, цель которых – помочь браузеру в работе с данными. Содержимое тега <head> не отображается на html-странице, за исключением тега <title> который устанавливает окна web-страницы;

– <body></body> — открывающий и закрывающий тэг, указывающий на начало и окончание основного блока html-документа, в котором будет содержаться ее контент;

– <h1></h1> — открывающий и закрывающий тэг, указывающий на начало и окончание заголовка. Всего таких тегов может быть 6, и отличаются они величиной шрифта - чем выше числовой порядок заголовка, тем меньшими буквами он будет отражаться. Вместе с тегом , <h1> формирует структуру самого контента: в частности метки определяют начало и конец абзацев текста.

Таким образом, один за другим формируются разные элементы, которые впоследствии будут представлены на выходной странице.

1.7 JavaScript

Javascript — это язык программирования, с помощью которого веб-страницам придается интерактивность. С его помощью создаются приложения, которые включаются в HTML-код, к примеру, анкеты или формы регистрации, которые заполняются пользователем или диаграммы и графики как представлено в данной работе. Часто Javascript путают с языком программирования Java, однако общего между ними очень мало. К тому же, некоторые сравнивают Javascript с языками Python, Self, Ruby. Однако это особенный язык, который существует сам по себе.

С помощью Javascript можно изменять страницу, изменять стили элементов, удалять или добавлять теги. С его помощью можно узнать о любых манипуляциях пользователя на странице (прокрутка страницы, нажатие любой клавиши, клики мышкой, увеличение или уменьшение рабочей области экрана

и т.д.). Через Javascript можно к любому элементу HTML-кода получить доступ и делать с этим элементом множество манипуляций. Можно загружать данные, не перезагружая страницу(Ajax), выводить сообщения, считывать или устанавливать cookie и выполнять множество других действий.

Язык Javascript уникален тем, что он поддерживается практически всеми браузерами и полностью интегрируется с ними, а все что можно сделать с его помощью – делается довольно просто. Простой пример использования Javascript — это вывод предупреждающего сообщения на экран браузера:

```
<form>
<input name=hallo type=submit value=”alert” onClick=”alert('Предупреждение')”>
</form>
```

1.8 Микросервес

Микросервис — архитектурный стиль при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и взаимодействуют с остальными используя легковесные механизмы, как правило HTTP. Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Сравним микросервисы с монолитом (monolithic style): приложением, построенном как единое целое. Корпоротивные приложения часто включают три основные части: пользовательский интерфейс (состоящий как правило из HTML страниц и javascript-а), база данных (как правило реляционной, со множеством таблиц) и сервер. Серверная часть обрабатывает HTTP запросы, выполняет доменную логику, запрашивает и обновляет данные в БД,

заполняет HTML страницы, которые затем отправляются браузеру клиента. Любое изменение в системе приводит к пере сборке и развертыванию новой версии серверной части приложения.

Монолитный сервер — довольно очевидный способ построения подобных систем. Вся логика по обработке запросов выполняется в единственном процессе, при этом вы можете использовать возможности вашего языка программирования для разделения приложения на классы, функции и namespace-ы. Вы можете запускать и тестировать приложение на машине разработчика и использовать стандартный процесс развертывания для проверки изменений перед выкладыванием их в продакшн. Вы можете масштабировать монолитное приложения горизонтально путем запуска нескольких физических серверов с балансировщиком нагрузки.

Монолитные приложения могут быть успешными, но все больше людей разочаровываются в них, особенно в свете того, что все больше приложений развертываются в облаке. Любые изменения, даже самые небольшие, требуют пересборки и развертывания всего монолита. С течением времени, становится труднее сохранять хорошую модульную структуру, изменения логики одного модуля имеют тенденцию влиять на код других модулей. Масштабировать приходится все приложение целиком, даже если это требуется только для одного модуля этого приложения.

Эти неудобства привели к архитектурному стилю микро сервисов: построению приложений в виде набора сервисов.

В дополнение к возможности независимого развертывания и масштабирования каждый сервис также получает четкую физическую границу, которая позволяет разным сервисам быть написанными на разных языках программирования. Они также могут разрабатываться разными командами.

Мы не утверждаем, что стиль микро сервисов это инновация. Его корни уходят далеко в прошлое, как минимум к принципам проектирования,

использованным в Unix. Но мы тем не менее считаем, что недостаточно людей принимают во внимание этот стиль и что многие приложения получат преимущества если начнут применять этот стиль.

1.8.1 Свойства архитектуры микро сервисов

Мы не можем сказать, что существует формальное определение стиля микро сервисов, но мы можем попытаться описать то, что мы считаем общими характеристиками приложений, использующих этот стиль. Не всегда они встречаются в одном приложении все сразу, но, как правило, каждое подобное приложение включает в себя большинство этих характеристик. Мы попробуем описать то, что мы видим в наших собственных разработках и в разработках известных нам команд.

1.8.2 Разбиение через сервисы

В течение всего срока нашего пребывания в индустрии мы видим желание строить системы путем соединения вместе различных компонент, во многом так же, как это происходит в реальном мире. За последние пару десятков лет мы видели большой рост набора библиотек, используемых в большинстве языков программирования.

Говоря о компонентах, мы сталкиваемся с трудностями определения того, что такое компонент. Наше определение такого: компонент — это единица программного обеспечения, которая может быть независимо заменена или обновлена.

Архитектура микро сервисов использует библиотеки, но их основной способ разбиения приложения — путем деления его на сервисы. Мы определяем библиотеки как компоненты, которые подключаются к программе и вызываются ею в том же процессе, в то время как сервисы — это

компоненты, выполняемые в отдельном процессе и коммуницирующие между собой через веб-запросы или remote procedure call (RPS).

Главная причина использования сервисов вместо библиотек — это независимое развертывание. Если вы разрабатываете приложение, состоящее из нескольких библиотек, работающих в одном процессе, любое изменение в этих библиотеках приводит к пере развёртыванию всего приложения. Но если ваше приложение разбито на несколько сервисов, то изменения, затрагивающие какой-либо из них, потребуют пере развёртывания только изменившегося сервиса. Конечно, какие-то изменения будут затрагивать интерфейсы, что, в свою очередь, потребует некоторой координации между разными сервисами, но цель хорошей архитектуры микро сервисов — минимизировать необходимость в такой координации путем установки правильных границ между микро сервисами, а также механизма эволюции контрактов сервисов.

Другое следствие использования сервисов как компонент — более явный интерфейс между ними. Большинство языков программирования не имеют хорошего механизма для объявления Published Interface. Часто только документация и дисциплина предотвращают нарушение инкапсуляции компонентов. Сервисы позволяют избежать этого через использование явного механизма удаленных вызовов.

Тем не менее, использование сервисов подобным образом имеет свои недостатки. Удаленные вызовы работают медленнее, чем вызовы в рамках процесса, и поэтому API должен быть менее детализированным (coarser-grained), что часто приводит к неудобству в использовании. Если вам нужно изменить набор ответственностей между компонентами, сделать это сложнее из-за того, что вам нужно пересекать границы процессов.

В первом приближении мы можем наблюдать, что сервисы соотносятся с процессами как один к одному. На самом деле сервис может содержать множество процессов, которые всегда будут разрабатываться и

развертываться совместно. Например, процесс приложения и процесс базы данных, которую использует только это приложение.

1.8.3 Размер микросервисов

Хотя термин «Микросервис» стал популярным названием для этого архитектурного стиля, само имя приводит к чрезмерному фокусу на размере сервисов и спорам о том, что означает приставка «микро». В наших разговорах с теми, кто занимался разбиением ПО на микро сервисы, мы видели разные размеры. Наибольший размер был у компаний, следовавших правилу «Команда двух пицц» (команда, которую можно накормить двумя пиццами), т.е. не более 12 человек (*прим.: следуя этому правилу, я в команде должен быть один*). В других компаниях мы видели команды, в которых шестеро человек поддерживали шесть сервисов.

Это приводит к вопросу о том, есть ли существенная разница в том, сколько человек должно работать на одном сервисе. На данный момент мы считаем, что оба этих подхода к построению команд (1 сервис на 12 человек и 1 сервис на 1 человека) подходят под описание микро сервисной архитектуры, но возможно мы изменим свое мнение в будущем. (*прим.: со времен статьи появилось множество других статей, развивающих эту тему; наиболее популярным сейчас считается мнение о том, что сервис должен быть настолько большим, чтобы он мог полностью «уместиться в голове разработчика», независимо от количества строк кода*).

1.9 Bootstrap

Bootstrap (также известен как Twitter Bootstrap) — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML и CSS шаблоны оформления для типографии, веб-форм, кнопок, меток,

блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Bootstrap использует самые современные наработки в области CSS и HTML, поэтому необходимо быть внимательным при поддержке старых браузеров.

1.9.1 Основные преимущества.

Основные преимущества Bootstrap 3:

– экономия времени — Bootstrap позволяет сэкономить время и усилия, используя шаблоны дизайна и классы, и сконцентрироваться на других разработках;

– гармоничный дизайн — все компоненты платформы Bootstrap используют единый стиль и шаблоны с помощью центральной библиотеки. Дизайн и макеты веб-страниц согласуются друг с другом;

– простота в использовании — платформа проста в использовании, пользователь с базовыми знаниями HTML и CSS может начать разработку с Twitter Bootstrap;

– совместимость с браузерами — Twitter Bootstrap совместим с Mozilla Firefox, Yandex Browser, Google Chrome, Safari, Internet Explorer, Microsoft Edge и Opera;

– скорость работы — благодаря множеству готовых элементов вёрстка с Bootstrap занимает значительно меньше времени;

– масштабируемость — добавление новых элементов не нарушает общую структуру;

– лёгкая настраиваемость — редактирование стилей производится путём создания новых css-правил, которые исполняются вместо стандартных;

– большое количество шаблонов;

– огромное сообщество разработчиков;

– широкая сфера применения — Bootstrap используется в создании тем для практически любой CMS (OpenCart, Prestashop, Magento, Joomla, Bitrix, WordPress и любые другие), в том числе для одностраничных приложений.

2 Раздел проектирование и разработка

2.1 Описание системы

Итак, как мы уже знаем система состоит из нескольких элементов которые показаны на этой схеме. Все, кроме сайта настроек и базы данных, написано на Python и в этой системе используется подход микросервисов это значит, что каждая даже небольшая задача выполняется своей отельной программой. Этот подход к программированию описан в главе 1.8. Также удобно дописывать разные дополнительные модули если это необходим.

Все взаимодействие в системе реализуются с помощью HTTP протокола за исключением Считывающего устройства и разбери.

Далее, представлена структурная схема электронной системы учёта посещений (рисунок 3).

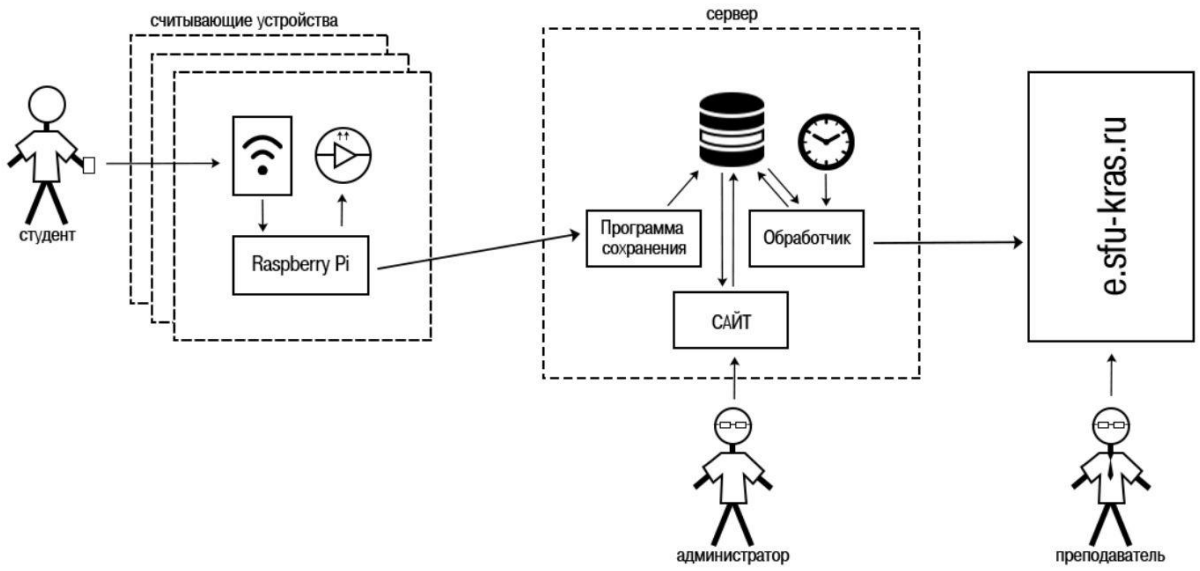


Рисунок 3 – Структурная схема

2.1.2 Использование HTTP глаголов для связи с интерфейсами и программами между собой.

HTTP глаголы составляют основную часть "единого интерфейса", ограничивающего и предоставляющего возможность осуществлять действия над существительным-ресурсом. Основными или наиболее часто используемыми HTTP глаголами (или методами, как их иногда называют) являются POST, GET, PUT, и DELETE. Они соответствуют операциям создания, чтения, обновления и удаления (или в совокупности - CRUD). Есть еще и другие глаголы, но они используются реже. Из реже используемых методов выделяются OPTIONS и HEAD. Мы использовали всего два метода POST и GET поэтому о я опишу их подробно.

2.1.2.1 GET

HTTP метод GET используется для получения (или чтения) представления ресурса. В случае “удачного” (или не содержащего ошибок) адреса, GET возвращается представление ресурса в формате XML или JSON в сочетании с кодом состояния HTTP 200 (OK). В случае наличия ошибок обычно возвращается код 404 (NOT FOUND) или 400 (BAD REQUEST).

В соответствии спецификации HTTP, GET (также как и HEAD) запросы используются только для чтения данных, не изменяя их. Таким образом, при соблюдении данного соглашения, они считаются безопасными. То есть они могут использоваться без риска изменения данных, вне зависимости от того, один раз данные были получены, или же 10, или ни разу вовсе. GET (а также HEAD) запросы являются идемпотентными (тождественными), что подразумевает получение идентичных данных при использовании одних и те же запросов (как при единичном обращении, так и при многократном).

2.1.2.2 POST

POST запрос наиболее часто используется для создания новых ресурсов. На практике он используется для создания вложенных ресурсов. Другими словами, при создании нового ресурса, POST запрос отправляется к родительскому ресурсу и, таким образом, сервис берет на себя ответственность на установление связи создаваемого ресурса с родительским ресурсом, назначение новому ресурсу ID и т.п.

При успешном создании ресурса возвращается HTTP код 201, а также в заголовке «Location» передается адрес созданного ресурса.

POST не является безопасным или идемпотентным запросом. Потому рекомендуется его использование для не идемпотентных запросов. В результате выполнения идентичных POST запросов предоставляются сильно похожие, но не идентичные данные.

2.1.3 Подключение Raspberry Pi и считывающего устройства

Модуль RFID-RC522 SainSmart работает с тегами Mifare RFID и использует чип RC522Для Raspberry Pi важно использовать модуль 3.3V так как он совместим с входами напряжения на Raspberry Pi.

Последовательной шиной периферийного интерфейса (SPI) является шина синхронной последовательный канал передачи данных, который Raspberry Pi поддерживает через GPIO, PI поддерживает два входных устройств с использованием CE (Chip Enable) выходов.

2.1.4 Включение SPI на Raspberry Pi

Поскольку SPI не включена по умолчанию, мне нужно было отредактировать raspi-blacklsit.conf для того, чтобы включить интерфейс SPI;

Необходимо было добавить '#' в начале строки spi-bcm2708 убрать его из черного списка. Сохраните файл, и перезагрузить Raspberry Pi, после чего команда lsmod должна показать ИПБ устройство (spi_bcm2708) включен.

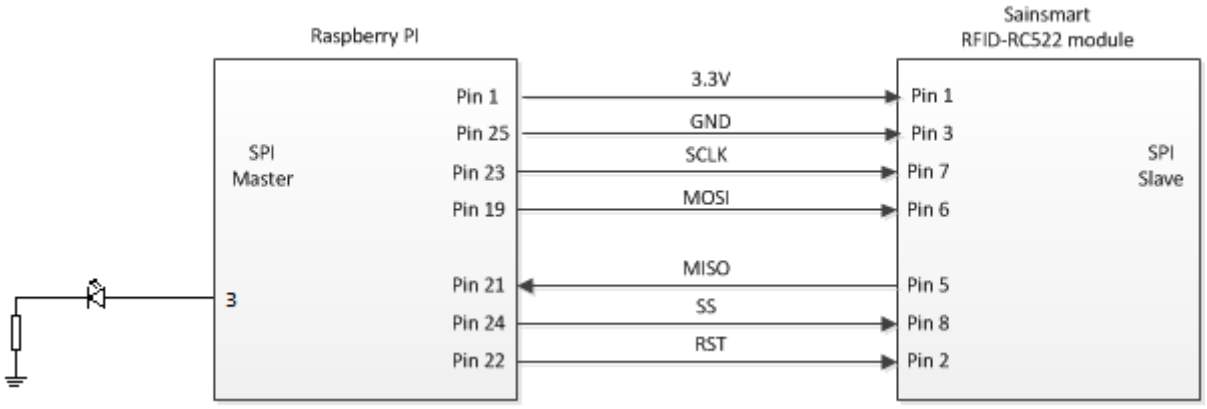


Рисунок 4 – Схема подключения

Чтобы использовать модуль из Python, необходимо загрузить библиотеку SPI, однако нам нужно установить ‘python-dev’, чтобы позволить нам установить оболочку SPI.

Чтобы установить ‘python-dev’ необходимо ввести в консоли:

sudo apt-get install python-dev

Для считывания данных с шины SPI в Python мне нужен набор подпрограмм, подходящий набор SPI-Py, я взяла с GitHub.

Также я использовала макетную плату для того чтобы включить в систему светодиод (рисунок 5).

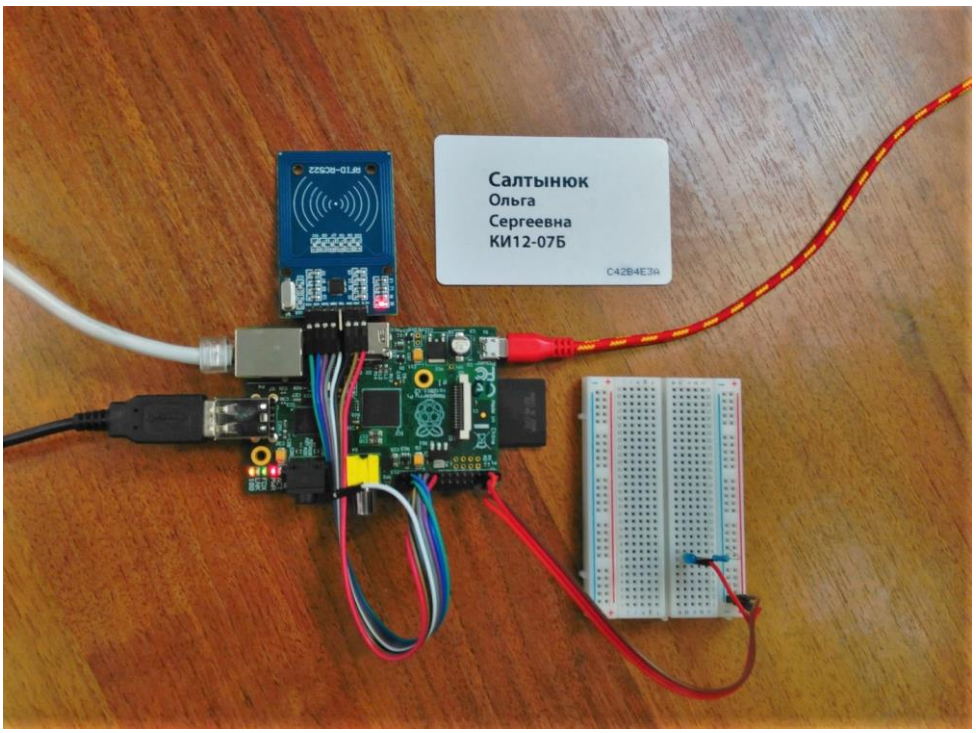


Рисунок 5 – Считывающее устройство и электронная карта

2.2 Сайт для внесения и редактирования данных.

Данный сайт был разработан с помощью веб-платформы на Java Script – Meteor.

Структура самого Meteor построена также на микросервисах. Далее, будет показано как это используется.

Как и в любой классический сайте, этот сайт написан на HTML, CSS, Java Script. С использованием технологий Bootstrap.

Но Java Script используется как в написание backend так и в написании frontend. Далее будет комментироваться весь код ,в общем, но также будут указываться места где backend, а где frontend.

В основном я хочу описать именно особенности написания сайта на Meteor. Так как все остальное достаточно однообразно.

Начнем с той части, которая описывает внешнюю часть нашей главной страницы.

```
<template name="home">
```

Первая строчка это template, таким образом, мы определяем часть кода программы, и даем имя, по которому мы можем к ней обращается в дальнейшем. Когда нам нужна эта часть кода в другом коде мы можем в некоторых случаях просто обратиться по имени как здесь.

```
FlowRouter.route('/home', {
    name: 'home',
    action: function() {
        BlazeLayout.render("mainLayout", {
            content: "home"
        });
    }
});
```

Или написав вот такую конструкцию.

```
{{> home}}
```

Эта часть кода написана на языке HTML, но в ней встречаются необычные конструкции `{{> studentList}}` и `{{> addStudentData}}` это значит, что вместо этих конструкций мы подставляем сюда код обозначенный `<template name=" studentList "> и <template name=" addStudentData ">`

```
<table class="table table-bordered table-condensed">
                    <caption>студенты </caption>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>MudleID</th>
                            <th>ФИО</th>
                            <th>Группа</th>
                            <th class="otstup">x</th>
                        </tr>
```

```
                    </thead>
                    <tbody>
                        {{> studentList}}

                    </tbody>
                </table>
                {{> addStudentData}}
            </div>
               </div>
            </div>
        </div>
```

Эта часть кода выводит данные пользователь, который зашёл на сайт. Это значит, что программа берет данные с сервера о юзере, который авторизовался и записывает их в эту форму, отображающую эти данные на странице.

```
    {{#with currentUser}}
        <div class="ui raised segment">
            <h1 class="ui header">Current User</h1>
            <p>First Name: {{profile.firstName}}</p>
            <p>Last Name: {{profile.lastName}}</p>
            <p>Profile Picture: <img class="ui small circular
image" src="{{profile.profPicture}}" /></p>
            <p>Organization: {{profile.organization}}</p>
        </div>
        {{/with}}
```

Рассмотрим серверную часть приложения, которая помогает создавать и хранить личные данные о пользователях в безопасности.

```
    Accounts.onCreateUser(function(options, user) {
```

Использовать предоставленный профиль в настройках, или создать пустой объект профиля.

```
user.profile = options.profile || {};
```

Назначает имя и фамилию вновь созданному объекту пользователя.

```
        user.profile.firstName = options.firstName;
        user.profile.lastName = options.lastName;
           user.profile.profPicture = Meteor.absoluteUrl() +
"img/default/user.jpg";
        user.profile.organization = ["Org"];
        user.roles = ["User"];
```

Возвращает объект пользователя.

```
        return user;
    });
```

Далее, рассмотрим окно авторизации и регистрации, так как они идентичны рассмотрим одно из них. Это HTML поэтому здесь все просто и понятно. Описаны окошки для в вода, input и кнопка button.

```
    <template name="register">
     <form id="register" class="login">
      <h1 class="ui header">Register</h1>
      <input class="email" type="email" placeholder="Email
Address" id="email" />
      <input class="text" type="text" placeholder="First Name"
id="first-name" />
      <input class="text" type="text" placeholder="Last Name"
id="last-name" />
```

```
        <input          class="password"          type="password"
placeholder="Password" id="password" />
        <button           class="button"          id="reg-button"
type="submit">Регистрация</button>
        </form>
      </template>
```

На рисунке 6 представлен внешний вид это части программы.

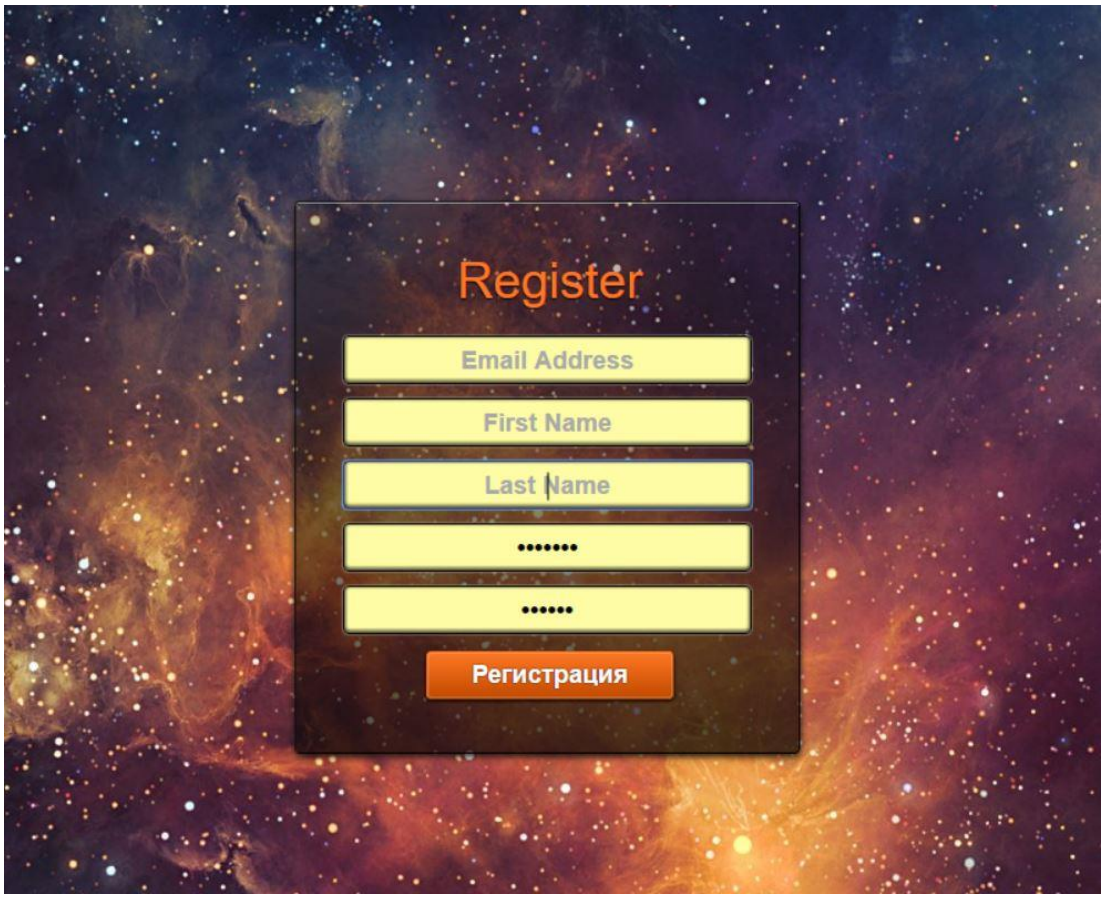


Рисунок 6 – Окно регистрации

А теперь обратимся к backend.

Первые 3 строки обработчик нажатия кнопки.

```
Template.register.events({
    'click #register-button': function(e, t) {
        e.preventDefault();
```

Получение значений полей ввода

```
var email = $('#email').val(),
    firstName = $('#first-name').val(),
    lastName = $('#last-name').val(),
    password = $('#password').val(),
    passwordAgain = $('#password-again').val();
```

Проверка пароля на длину не менее 6 символов

```
var isValidPassword = function(pwd, pwd2) {
    if (pwd === pwd2) {
        return pwd.length >= 6 ? true : false;
    } else {
        return swal({
            title: "Passwords don't match",
            text: "Please try again",
            showConfirmButton: true,
            type: "error"
        });
    }
}
```

Если проверка пройдена ставим отметку, иначе нужно ввести другой пароль.

Далее идет функция Meteor.loginWithPassword().

```
if (isValidPassword(password, passwordAgain)) {
    Accounts.createUser({
        email: email,
        firstName: firstName,
        lastName: lastName,
        password: password
```

```
        }, function(error) {
            if (error) {
                return swal({
                title: error.reason,
                text: "Please try again",
                showConfirmButton: true,
                type: "error"
            });
            } else {
                FlowRouter.go('/');
             }
        });
     }

     return false;
    }
});
```

Маршрутизация на сайте осуществляется с помощью пакета Iron Router.

Iron Router – это пакет маршрутизации, который был задуман специально для Meteor приложений.

Он не только помогает в маршрутизации (настройке путей), но может и позаботиться о фильтрации (сопоставлении действий и некоторых путей), и даже управлять подписками (контролировать, какой путь имеет доступ к этим данным).

Единственное что необходимо сделать это написать адрес страницы, точнее, её имя шаблона во второй и четвёртой строке.

```
FlowRouter.route('/', {
    name: 'loginpromt',
    action: function() {
        BlazeLayout.render("mainLayout", {
            content: "login"
```

```
        });
    }
});

FlowRouter.route('/home', {
    name: 'home',
    action: function() {
        BlazeLayout.render("mainLayout", {
            content: "home"
        });
    }
});

FlowRouter.route('/register', {
    name: 'register',
    action: function() {
        BlazeLayout.render("mainLayout", {
            content: "register"
        });
    }
});
```

И также у нас есть та часть программы которая отвечает за сохранность наших данных. То есть не выдает и не разрешает редактировать данные пользователям у которых нет прав доступа.

```
import { Meteor } from 'meteor/meteor';

Meteor.startup(() => {
  // code to run on server at startup
});

Meteor.methods({
  'addStudentData'(studentData) {
```

```
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Students.insert(studentData);
},
'addRaspisanieData'(raspisanieData) {
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Raspisanie.insert(raspisanieData);
},
'updateStudentData'(id, studentData) {
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Students.update(id, studentData);
},
  'updateRaspisanieData'(id, raspisanieData) {
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Raspisanie.update(id, raspisanieData);
},

'deleteStudent'(id){
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Students.remove(id);
},
  'deleteRaspisanie'(id){
  if (! Meteor.userId()) {
    throw new Meteor.Error('not-authorized');
  }
  Raspisanie.remove(id);
```

```
  },
  'getStudent'(id){
    if (! Meteor.userId()) {
      throw new Meteor.Error('not-authorized');
    }
    return Students.findOne(id);
  },
     'getRaspisanie'(id){
    if (! Meteor.userId()) {
      throw new Meteor.Error('not-authorized');
    }
    return Raspisanie.findOne(id);
  },
});
```

Таким образом из базы данных данные попадают на страницу.

`{{#each students}}` это цикл.

```
<template name="studentList">
    {{#each students}}
      <tr>{{> studentData}}</tr>
    {{/each}}
</template>
```

А это backend этого же действия. `return Students.find();` это обращение к базе студентов.

```
Template.studentList.helpers({
  students: function() {
    return Students.find();
  }
});
```

Таким образом происходит заполнение таблицы.

```
<template name="studentData">
     <td>{{Id}}</td>
     <td>{{MoodleId}}</td>
     <td>{{FIO}}</td>
     <td>{{Group}}</td>
    <td>{{> task}} {{> Sedit}}</td>
</template>
```

А вот так это выглядит в графическом интерфейсе (Рисунок 7).

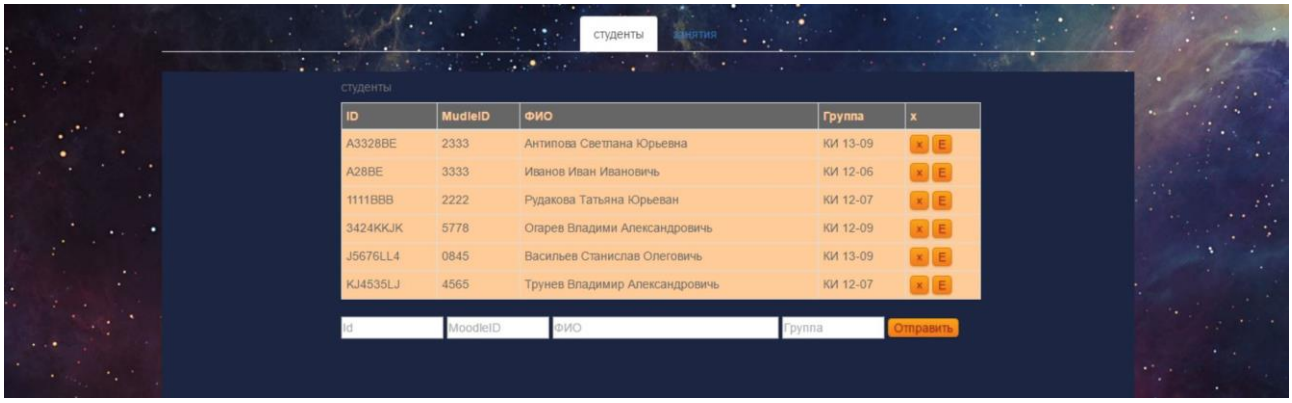


Рисунок 7 – Таблица студентов

Далее рассмотрим функции удаление и редактирования записи. У каждой записи есть кнопка удаления.

```
<template name="task">
    <button class="delete">&times;</button>
</template>
```

У Метеор есть специальные функции для удаления вот так она выглядит.

```
Template.task.events({
  'click .delete'() {
     Meteor.call("deleteStudent", this._id);
    Students.remove(this._id);
```

```
        Meteor.call("deleteRaspisanie", this._id);
      Raspisanie.remove(this._id);
    },
  });
```

При нажатии кнопки редактировать, запись добавляется в окна, внизу таблицы. После изменения записей нужно нажать кнопку сохранить, и они сохранятся в новом виде.

```
  Template.Redit.events({
    'click .Redit'() {

          console.log('Redit button click');
          id = this._id;
        raspisanieData = Raspisanie.findOne(id);
          $("#secretId1")[0].value = raspisanieData._id;
          $(".str5")[0].value = raspisanieData.cab;
          $(".str6")[0].value = raspisanieData.time;
                  $(".str7")[0].value                    =
raspisanieData.weak;
                  $(".str8")[0].value = raspisanieData.day;
          $(".str9")[0].value = raspisanieData.predmet;
          $(".str10")[0].value = raspisanieData.IDpredmet;

    },
  });
```

Одна из главных функций добавление данных в базу данных производится таким образом.

```
  Template.addStudentData.events({
    'submit .add-student'(event)
```

Объявление переменных.

```
        event.preventDefault();
        const target = event.target;
        const secretId = target.secretId.value;
        const studentData = {
          Id : target.Id.value,
          MoodleId : target.MoodleId.value,
          FIO : target.FIO.value,
          Group : target.Group.value,
        }
```

Хочу обратить внимание что если мы не зарегистрированы в системы то мы не можем добавить, удалить или отредактировать данные.

```
        if(secretId !== '')
```

Добавление в базу данных.

```
          Meteor.call("updateStudentData",
secretId,studentData);
        else
          Meteor.call("addStudentData",studentData);
```

Очищаем формы.

```
        target.Id.value = '';
          target.MoodleId.value = '';
          target.FIO.value = '';
          target.Group.value = '';
          target.secretId.value = '';
      },
```

2.3 Обработчик данных

Обработка данных в системе реализуется с помощью трех программ, одна находится на Raspberry Pi и отправляет данные на сервер.

Вторая находится на сервере и принимает данные которые приходят со считывающего устройства и отправляет их в базу данных. Третья часть производит обработку посещений. То есть она берет уже существующие данные о студентах и сравнивает с теми студентами, кто пришел и отметился. Далее отправляет Post запрос на e.sfu-kras.ru. Все эти программы между собой общаются с помощью HTTP запросов.

Далее я более подробно разъясню что происходит внутри каждой программы.

2.3.1 Первая программа reader.py

Подключение библиотек для работы со считывающем устройством и для подключение его к Raspberry Pi

```
import MFRC522
import RPi.GPIO as GPIO
from time import sleep, time
from requests import post
```

Задает номер выхода для светодиода, Задаём адрес для отправки данных, номер устройства или кабинета у которого установлено устройство.

```
PIN = 3
URL = 'http://10.3.73.105/id'
CAB = '601'
reader = MFRC522.MFRC522()
```

Функции для назначение входов и выходов на Raspberry Pi.

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(pin, GPIO.OUT)
```

Функция для считывание ID с карты.

```
def read_uid():
    status,TagType=
reader.MFRC522_Request(reader.PICC_REQIDL)
    status, uid = reader.MFRC522_Anticoll()
    print(status, uid)
    return status, uid
```

Функция для свечения светодиода.

```
def blink():
     GPIO.output(pin, True)
     sleep(.3)
     GPIO.output(pin, False)
```

Главный цикл: считываем; смотрим что действительно пришло; мигаем светодиодом; переводим ID в нужный формат; отправляем post запрос; ждем одну секунду.

```
while True:
    status,id = read_uid()
    if status == 0:
     blink()
     sid = "{0:X}{1:X}{2:X}{3:X}".format(*id)
     post(URL, {"cabinet": CAB, "id" : sid, "time" : time()})
    sleep(1)
```

2.3.2 Вторая программа reciver.py

В этой программе мы принимаем ту информацию, которую отправила нам первая программа и сохраняем ее в нашу базу данных. Для этого подключаем библиотеку для работы с MongoDB и пользуясь ее методами отправляем наши данные в MongoClient("localhost:3001")

```
from pymongo import MongoClient

client = MongoClient("localhost:3001")
db = client['meteor']

@app.route('/id', methods=['POST'])
def moodle_test():
     print(request.form)
     db.idtable.insert({"id": request.form["id"][:-2],

          "cabinet": request.form["cabinet"], "time" :
request.form["time"]})
```

2.3.3 Третья программа process.py

Далее я разъясню и прокомментирую некоторые части кода третьей части.

Подключение библиотеки для работы с Mongo и для работы со временем, открытия сессии для обмена запросами.

```
from pymongo import MongoClient

from requests import session
import re
import time
```

Переменные для работы с базой данных с разными таблицами.

```
dbs = MongoClient("localhost:3001")
meteor_db =dbs['meteor']
idtable = meteor_db['idtable']
students = meteor_db['students']
raspisanie = meteor_db['raspisanie']
```

Переменна для авторизации на e.sfu-kras.ru.

```
payload = {
    'username': '******',
    'password': '******',
    'rememberusername': 1
}
```

Функция для передачи и обработки данных по времени блок схема работы функции подставлена ниже (рисунок 8).

```
def process(para_time):
    with session() as c:
        r = c.post('https://e.sfu-kras.ru/login/index.php', data=payload)
        response = c.get('https://e.sfu-kras.ru/grade/report/grader/index.php?id=1029')
        res = re.search('sesskey":"(\w*)"', response.text)
        sesskey = res.group(1)
        main(para_time, sesskey, c)
```

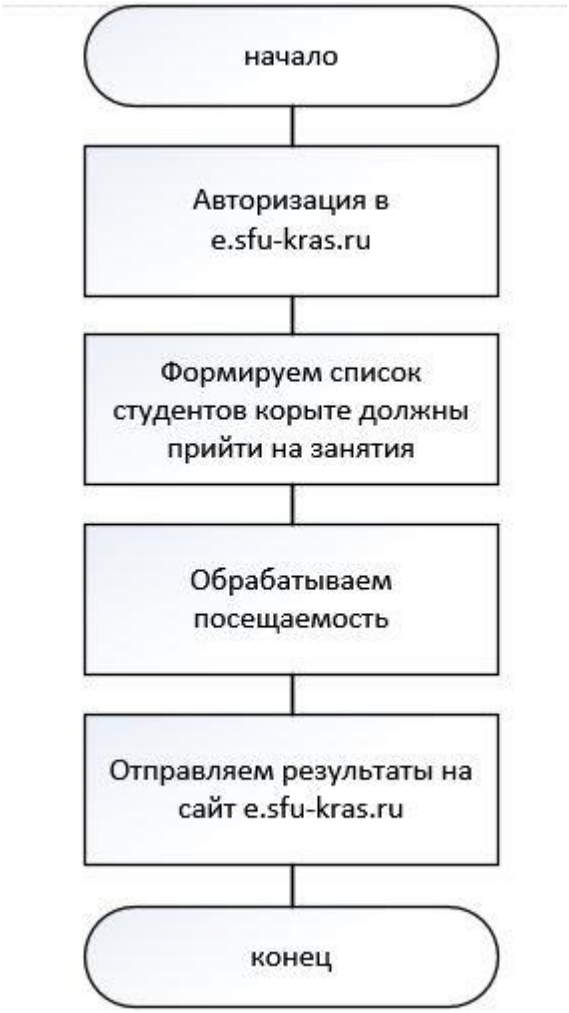


Рисунок 8 – Блок схема программы 3

Основная функция для формирования списка студентов которые должны быть на занятии.

```
    def main(para_time, sesskey, c):
        for predmet in get_predmet(para_time):
            print("== predmet: \n" + str(predmet))
            group_list = get_all_group(predmet)
            print("=== groups: \n" + str(group_list))
            students_list = list(get_all_student(group_list))
            print("===        students_list:        \n"        +
str(list(students_list)))
```

```
            zanyatie = int(get_zanyatie(predmet))
            print("=== zanyatie: \n" + str(zanyatie) + " isTrue
" + str(bool(zanyatie)))
            if bool(zanyatie):
                print("= process Attandance")
                process_attandance(students_list,       predmet,
para_time, zanyatie, sesskey, c)

        idtable.remove({})
```

Функция сравнивает список пришедших студентов, и тех кто должен прийти, и по результатам сравнений формирует запрос.

```
    def  process_attandance(students_list,  predmet,  para_time,
zanyatie, sesskey, c):
        print("== in process_attandance")
        print("== students_list: \n" + str(list(students_list)))
        for student in students_list:
            print("student " + str(student))
            cabinet = get_cabinet(predmet)
            if_attended = attend(student, para_time, cabinet)
            mark(if_attended,  predmet["IDpredmet"],  zanyatie,
student["MoodleId"], sesskey, c)
```

Функция для поиска списка предметов в текущее время.

```
    def get_predmet(para_time):
        print(list(raspisanie.find()))
        return raspisanie.find({"time": str(para_time)})
```

Функция для поиска списка групп которые должны прийти на предмет.

```
    def get_all_group(predmet):
```

```
    predmet_str =  predmet["group"]
    group_list = predmet_str.split(',')
    return [g.strip() for g in group_list]
```

Функция которая формирует список из всех студентов из этих групп.

```
def get_all_student(group_list):
    query = [ {"Group" : g } for g in group_list]
    return students.find({ '$or' : query})
```

Функция которая возвращает кабинет в кортом проходит занятие.

```
def get_cabinet(predmet):
    return  predmet["cab"]
```

Функция которая формирует список всех занятий предмета и потом удаляет занятие которое прошло.

```
def get_zanyatie(predmet):
    zanyatie_list = predmet["zanyatie"]
    if zanyatie_list:
        cur_zanyatie = zanyatie_list.pop()
        raspisanie.update({"_id":predmet["_id"]},  {"$set":{
"zanyatie" : zanyatie_list}})
        return cur_zanyatie
    else:
        return 0
```

Функция проверяет пришел ли студент на занятия в нужное время.

```
def attend(student, para_time, cabinet):
    attend_data  =  idtable.find_one({"id":  student["Id"],
"cabinet" :cabinet})
    if attend_data:
```

```
            return is_correct_time(para_time,
attend_data["time"])
        else:
            return False
```

Формирование запроса для сайта e.sfu-kras.ru.

```
    def mark(if_attended, predmetId, zanyatieId, studentId,
sesskey, c):
        attend_data = 2 if if_attended else 1

        grade_data = {
         'id': predmetId,       #   // id курсаh
         'userid': studentId,   #  // id студента
         'itemid': zanyatieId,  # // id занятия
         'action':'update',
         'newvalue': attend_data ,    # // 1 посетил, 2 не_посетил
         'type':'scale',
         'sesskey': sesskey #  // M.cfg.sesskey
        }
        print("mark data: " + str(grade_data))
```

Отправка запроса.

```
    response=c.post('https://e.sfu-
kras.ru/grade/report/grader/ajax_callbacks.php', data=grade_data)
```

Функция для перевода времени в нужный формат.

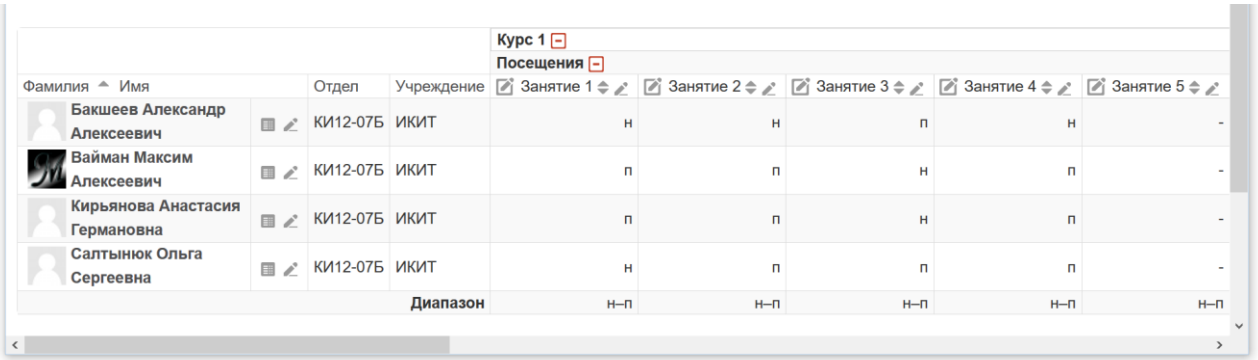
```
    def get_para_time(cur_time):
        hour = time.localtime(cur_time).tm_hour

        if hour == 8:
            return 1
        elif hour == 10:
```

```
        return 2
    elif hour == 12:
        return 3
    elif hour == 14:
        return 4
    elif hour == 15:
        return 5
    elif hour == 17:
        return 6
    elif hour == 19:
        retur
```

2.4 Тестирование системы

Было проведено тестирование системы, результаты тестирование представлены ниже (рисунок 9). Каждый компонент системы работает и справно и выполняет поставленные задачи. Также вся система в общем отработала без ошибок и выполнила поставленную задачу.



			Курс 1				
			Посещения				
Фамилия ▲ Имя	Отдел	Учреждение	Занятие 1	Занятие 2	Занятие 3	Занятие 4	Занятие 5
Бакшеев Александр Алексеевич	КИ12-07Б	ИКИТ	н	н	п	н	-
Вайман Максим Алексеевич	КИ12-07Б	ИКИТ	п	п	н	п	-
Кирьянова Анастасия Германовна	КИ12-07Б	ИКИТ	п	п	н	п	-
Салтынюк Ольга Сергеевна	КИ12-07Б	ИКИТ	н	п	п	п	-
		Диапазон	н–п	н–п	н–п	н–п	н–п

Рисунок 9 – Таблица посещений на e.sfu-kras.ru

ЗАКЛЮЧЕНИЕ

В заключение могу сказать, что проектируемая система была реализована и работает исправно. Спроектировано считывающее устройство. Разработан сайт для обработки данных. Создана программа для корректной обработки данных, данные отправляются и отображаются на сайт e.sfu-kras.ru

Все блоки этой системы были разработаны и написаны на разных языках и все вместе работает исправно работают исправно. Так как мы достигли своей цели.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Дронов В.А. Программирование. — СПб.: БХВ-Петербург, 2006 — 706 с.: ил.
2. Документация Bootstrap шаблоны [Электронный ресурс]. — Режим доступа: http://bootstrap-3.ru/bootstraptheme.php.
3. Уроки по Bootstrap [Электронный ресурс]. — Режим доступа: http://dedushka.org/uroki/6901.html.
4. Отличительные особенности JavaScript [Электронный ресурс]. — Режим доступа: http://articles-hosting.ru/329/javascript-%E2%80%93-chto-eto-takoe.html.
5. Особенности HTML [Электронный ресурс]. — Режим доступа: http://lpgenerator.ru/blog/2013/10/21/chto-takoe-html-korotko-o-glavnom.
6. Документация Meteor [Электронный ресурс]. — Режим доступа: http://ru.discovermeteor.com/.
7. Руководство пользование Mongo db [Электронный ресурс]. — Режим доступа: http://metanit.com/nosql/mongodb/.
8. Документация Python [Электронный ресурс]. — Режим доступа: https://www.python.org/doc/.
9. Плюссы микросервесной архитектуры [Электронный ресурс]. — Режим доступа: https://habrahabr.ru/post/261237/.
10. Технология RFID [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/RFID.
11. Использование HTTP методов для создания RESTful сервисов [Электронный ресурс]. — Режим доступа: http://www.restapitutorial.ru/lessons/httpmethods.html.
12. SainSmart RFID-RC522 & Pi [Электронный ресурс]. — Режим доступа: http://geraintw.blogspot.ru/2014/01/rfid-and-raspberry-pi.html.

ПРИЛОЖЕНИЕ А

К пояснительной записки прилагается видео с демонстрацией работы систему продолжительностью 2мин. 30 сек.